

Gambling in a Computationally Expensive Casino: Algorithm Selection as a Bandit Problem

Matteo Gagliolo^{1,2} Jürgen Schmidhuber^{1,2,3}

¹ *IDSIA*, Lugano, Switzerland

² University of Lugano, Faculty of Informatics, Switzerland

³ TU, Munich, Germany

On-line Trading of Exploration and Exploitation
NIPS 2006 Workshop

Abstract

Automating algorithm selection and parameter tuning is an old dream of the AI community, which has been brought closer to reality in the last decade. Most available techniques are either *oblivious*, with no knowledge transfer across different problems; or are based on a model of algorithm performance, learned in a separate *offline* training sequence, which is often prohibitively expensive. We describe recent work in which the problem is treated in a fully *online* setting. A model of algorithm performance can be learned *and* used to reduce the cost of learning it. The resulting *exploration-exploitation* trade-off can be treated in the context of Bandit problems.

Keywords: algorithm selection, algorithm portfolios, online learning, life-long learning, bandit problem, adversarial setting, expert advice, survival analysis, randomized algorithms, restart strategies, satisfiability, constraint programming, performance modeling.

Algorithm selection

- ▶ Originated by (Rice '76)
- ▶ Meta-learning (Vilalta '02)
- ▶ Algorithm portfolios (Huberman '97)
- ▶ Empirical Hardness Models (Leyton-Brown et al. '02)

Core idea: use a model of algorithm performance to select the best of a finite set of algorithms $\{a_1, a_2, \dots, a_K\}$, for a given problem instance (*per instance* algorithm selection)

Offline vs. Online AS

Decision problems (e.g. SAT): the objective is to minimize solution **time**. A *model* of algorithm performance has to be learned. The model is used to predict performance of each algorithm on an unseen problem instance.

Offline learning: a set of *training problems* is given. Each training problem is solved with each algorithm. A model is then learned based on collected performance data (runtime values).

Issues: high computational cost.

Online learning: incoming problems are solved sequentially. The model is updated each time a problem is solved, and used to perform AS for the next problem instance.

Issues: Exploration vs. exploitation trade-off.

Exploration-Exploitation Trade-off in Online Algorithm Selection

Exploration: of the performances of the a_k on different problem instances.

Exploitation: of the best algorithm/problem combinations, based on current model's predictions.

Can I represent this trade-off as a bandit problem?

Related work: Max K-Armed Bandit Problem, for *optimisation* problems (maximize performance quality) (Cicirello et al. '05).

First game

- ▶ K algorithms $\{a_1, \dots, a_K\}$ are the K arms
- ▶ $\mathbf{p} \in [0, 1]^K$, $\sum_k p_k = 1$
- ▶ Algorithm a_k has a finite runtime $t_k(j)$ on problem b_j
- ▶ For each problem b_j
 - ▶ pick algorithm k with prob. p_k
 - ▶ run it
 - ▶ observe reward $1/t_k(j)$ if problem solved, 0 if not solved
 - ▶ update \mathbf{p} using your favorite bandit problem solver

This *partial information* game implements **per set** AS: in the limit, the algorithm that performs best on all problems is selected. In practice this can be much less efficient than **per instance** AS.

Second game

- ▶ K algorithms $\{a_1, \dots, a_K\}$ are the K arms
- ▶ $\mathbf{p} \in [0, 1]^K$, $\sum_k p_k = 1$
- ▶ Algorithm a_k has (possibly ∞) runtime $t_k(j)$ on problem b_j
- ▶ For each problem b_j
 - ▶ pick algorithm k with prob. p_k
 - ▶ run it for a short time δt , then pause it
 - ▶ observe reward $1/t_k(j)$ if problem solved, 0 if not solved
 - ▶ update \mathbf{p} using your favorite bandit problem solver

This partial info game also implements **per set** AS. It falls in the *non-oblivious* adversarial setting (Auer et al. '95): $t_k(j)$ is the *total* execution time of alg. a_k on problem b_j . The reward observed depends on the number of pulls of arm k during the solution of the current problem.

Third game

Consider the second game, with $\delta t \rightarrow 0$

- ▶ K algorithms $\{a_1, \dots, a_K\}$ are the K arms
- ▶ $\mathbf{s} \in [0, 1]^K$, $\sum_k s_k = 1$
- ▶ Algorithm a_k has (possibly ∞) runtime $t_k(j)$ on problem b_j
- ▶ For each problem b_j
 - ▶ run all a_k in parallel, sharing machine time $\propto \mathbf{s}$
 - ▶ observe reward $1/t_k(j)$ if problem solved, 0 if not solved
 - ▶ update \mathbf{s} using your favorite bandit problem solver

This partial info game also implements **per set** AS. Equivalent to a static *algorithm portfolio*.

Fourth game

Time allocator (TA): device that maps each problem instance to a share value \mathbf{s} , possibly based on a model of performance learned on previous instances. A TA can perform per instance selection.

- ▶ K algorithms $\{a_1, \dots, a_K\}$ are the K arms
- ▶ N time allocators $\{TA^{(1)}, \dots, TA^{(N)}\}$ are the N *experts*
- ▶ $\mathbf{p} \in [0, 1]^N, \sum_n p_n = 1$
- ▶ Algorithm a_k has (possibly ∞) runtime $t_k(j)$ on problem b_j
- ▶ For each problem b_j
 - ▶ run all a_k in parallel, sharing machine time $\propto \mathbf{s} = \sum_n p_n \mathbf{s}^{(n)}$
 - ▶ observe reward $1/t_k(j)$ if problem solved, 0 if not solved
 - ▶ update \mathbf{p} using your favorite solver for the *bandit problem with expert advice*

This game implements **per set** selection of the best time allocator, i.e., the TA that is best at performing per instance selection...

Learning Restart Strategies (IJCAI '07)

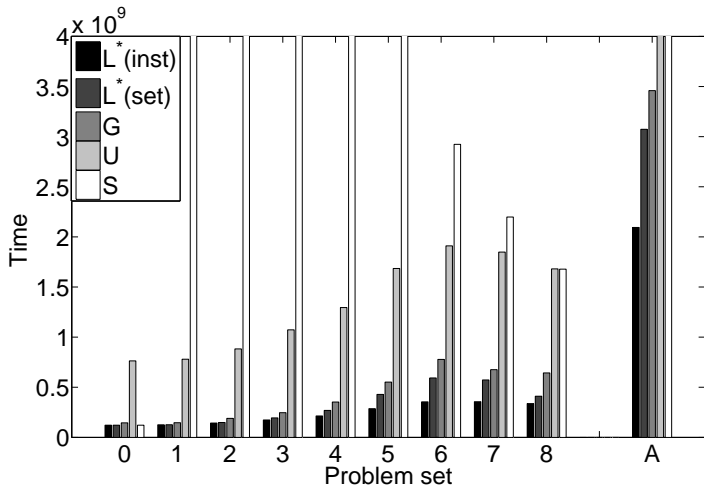
An example of the first game: GAMBLER .

BPS: EXP3 (Auer et al. '95).

- ▶ a_1 : randomized complete SAT solver (Gomes et al. '00), with universal restart strategy (Luby et al. '93).
- ▶ a_2 : same solver with model-based restart strategy. The model is updated as problems are solved.

9 sets of SAT-encoded graph coloring problems (Gent et. al. '99).

Learning Restart Strategies (IJCAI '07)



Time to solve each set of problems ($10^9 \approx 1$ min.). Per instance ($L^*(inst)$) and per set ($L^*(set)$) optimal restart strategies, GAMBLER (G), universal strategy (U), and Satz-Rand without restarts (S).

Learning Dynamic Algorithm Portfolios (AMAI '07)

An example of the fourth game: GAMBLETA .

BPS: EXP4 (Auer et al. '95).

- ▶ a_1 : randomized complete SAT solver (Gomes et al. '00).
- ▶ a_2 : local search SAT solver (Li et al '05).

SAT-UNSAT problems at phase transition ($uf-*$, $uu-*$ from SATLIB).

- ▶ a_1 : CASS.
- ▶ a_2 : CPLEX.

Auction Winner Determination Problem (WDP) benchmarks from (Leyton-Brown et al. '02).

Learning Dynamic Algorithm Portfolios (AMAI '07)

SAT-UNSAT $10^9 \approx 1$ min.	GAMBLETA	$2.88 \times 10^{10} \pm 1.06 \times 10^8$
	ORACLE	$2.53 \times 10^{10} \pm 5.17 \times 10^7$
	CUMOVH	0.138 ± 0.00324
WDP seconds	GAMBLETA	$1.12 \times 10^8 \pm 1.81 \times 10^5$
	ORACLE	$1.08 \times 10^8 \pm 7.61 \times 10^{-8}$
	CUMOVH	0.0381 ± 0.00167

Performance of GAMBLETA , and its overhead (CumOvh) compared to the best per instance selection (Oracle).

WDP comparison term (Leyton-Brown et al. '02): cumulative overhead 0.08 after *years* of training.

Conclusion

- ▶ GAMBLER : bound on regret w.r.t. the **per set** best restart strategy. K strategies: regret scaling $O(\sqrt{K \ln K})$.
- ▶ GAMBLETA : bound on regret w.r.t. the **per set** best time allocator: nothing can be guaranteed about the performance overhead on the *per instance* best algorithm. K algorithms, N time allocators: regret scaling $O(\sqrt{K \ln N})$.

Bibliography (<http://www.idsia.ch/~matteo/>)

[1] Gagliolo, M. Schmidhuber, J., Learning restart strategies. *IJCAI 2007*.

[2] Gagliolo, M., Schmidhuber, J., Learning Dynamic Algorithm Portfolios. *AI & MATH 2006 Special Issue of the Annals of Mathematics and Artificial Intelligence*. To appear.

This work was funded by SNF under grant 200020-107590/1